Exploring the java.util Package

Subtitle: Examples and Applications
Presented by: [Santhosh Kiran J]

Introduction to java.util Package

- Overview of java.util package
- Importance of utility classes in Java
- Key functionalities:
- Data structures, Collections Framework, Date and Time, and more

Collections Framework

- Overview of Collections Framework
- Key interfaces: List, Set, Queue, Map
- Importance of Collections for data manipulation

java.util Package Interfaces

Interface	Description
Collections	Root interface for the Collections API
Comparator	Provides sorting logic
Deque	Implements Deque data structure
Enumeration	Produces enumeration objects
Iterator	Provides iteration logic
List	Implements an ordered collection (Sequence)
Map	Implements Map data structure (key-value pairs)
Queue	Implements Queue data structure
Set	Produces a collection that contains no duplicate elements
SortedSet	Produces a set that remembers the total ordering

java.util Package Classes

Class	Description
Collections	Provides methods to represent and manage collections
Formatter	An interpreter for format strings
Scanner	Text scanner used to take user inputs
Arrays	Provides methods for array manipulation
LinkedList	Implements Doubly-linked list data structure
HashMap	Implements Map data structure using Hash tables
TreeMap	Implements Red-Black tree data structure
Stack	Implements last-in-first-out (LIFO) data structure
PriorityQueue	Implements unbounded priority queue data structure
Date	Represents a specific instance of time
Calendar	Provides methods to manipulate calendar fields
Random	Used to generate a stream of pseudorandom numbers.
StringTokenizer	Used to break a string into tokens
Timer, TimerTask	Used by threads to schedule tasks
UUID	Represents an immutable universally unique identifier

java.util Package Exceptions

Exception Class	Description
ConcurrentModificationException	When an object is modified concurrently without permission
EmptyStackException	Indicates that the Stack is empty
InputMismatchException	When the user does not provide valid input
MissingResourceException	Indicates absence of a resource
NoSuchElementException	Indicates absence of requested element
PatternSyntaxException	Indicates syntax error in a regular-expression pattern
IllegalFormatException	Indicates that a format string contains illegal syntax

Example - Using ArrayList

1. Create a dynamic array of strings
2. Add elements to the list
3. Retrieve and display elements

~~~~
\`\`\`java
import java.util.ArrayList;
import java.util.List;
public class ArrayListExample {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");
        System.out.println("Fruits: " + fruits);
    }
}
\`\`\`
~~~~

Example - Using HashMap

1. Create a mapping between student names and grades
2. Add entries to the map
3. Retrieve and display entries

```
\`\`\`java
import java.util.HashMap;
import java.util.Map;
public class HashMapExample {
    public static void main(String[] args) {
        Map<String, Integer> studentGrades = new HashMap<>();
        studentGrades.put("Alice", 90);
        studentGrades.put("Bob", 85);
        studentGrades.put("Charlie", 92);

        System.out.println("Student Grades: " + studentGrades);
    }
}
\`\`\`
```

Date and Time with java.util.Date

1. Create and display the current date

2. Format the date using SimpleDateFormat

~~~
\`\`\`java
import java.util.Date;
import java.text.SimpleDateFormat;
public class DateExample {
    public static void main(String[] args) {
        Date now = new Date();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        System.out.println("Current Date and Time: " + sdf.format(now));
    }
}
\`\`\`
~~~

Random Number Generation

1. Generate random integers and doubles

2. Use Random class for various use cases

```
\`\`\`java
import java.util.Random;
public class RandomExample {
    public static void main(String[] args) {
        Random random = new Random();
        int randomInt = random.nextInt(100); // Generate a random integer between 0 and 99
        double randomDouble = random.nextDouble(); // Generate a random double between 0.0 and 1.0
        System.out.println("Random Integer: " + randomInt);
        System.out.println("Random Double: " + randomDouble);
    }
}
\`\`\`
```

Working with Properties

1. Load and read properties from a file

2. Use Properties class to manage application settings

````
\`\`\`java
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
public class PropertiesExample {
    public static void main(String[] args) {
        Properties properties = new Properties();
        try (FileInputStream fis = new FileInputStream("config.properties")) {
            properties.load(fis);
            String username = properties.getProperty("username");
            String password = properties.getProperty("password");
            System.out.println("Username: " + username);
            System.out.println("Password: " + password);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}\`\`\`
````

Interview Questions

1. What is the difference between `ArrayList` and `LinkedList`?
2. How does `HashMap` handle collisions?
3. Explain the usage of `Collections` utility class.
4. How would you generate a random number between 1 and 10 using the `Random` class?
5. Describe a scenario where you would use the `Properties` class.

Difference between **ArrayList** and **LinkedList**

ArrayList and LinkedList are two different implementations of the List interface in Java, each with distinct characteristics.

•Storage Mechanism: ArrayList uses a dynamic array to store its elements, allowing for fast random access due to its contiguous memory allocation. In contrast, LinkedList uses a doubly linked list, where each element is a node containing references to the next and previous nodes, which enables efficient insertions and deletions.

•Performance: ArrayList offers better performance for retrieving elements (O(1) time complexity) but is slower for insertions and deletions (O(n) time complexity), especially in the middle of the list, since it may require shifting elements. On the other hand, LinkedList is faster for insertion and deletion operations (O(1) if the node is known) but slower for accessing elements (O(n) time complexity) because it may need to traverse from the beginning or end of the list.

•Memory Usage: ArrayLists generally consume less memory per element because they don't keep references for previous and next nodes, while LinkedLists require additional storage for these node references.

How does `HashMap` handle collisions?

In Java, a HashMap handles collisions using a technique called chaining:

When a collision occurs

- When two or more keys have the same hash code, they are mapped to the same bucket index in the HashMap.

Chaining

- The HashMap stores the collided elements in a linked list at the same index. This forms a chain of new entries in the linked list or tree structure at each index.
- Chaining ensures that all key-value pairs are stored, but it can make it difficult to efficiently search for a specific key in the chain.
- Java 8 introduced improvements to the HashMap implementation to reduce contention and improve performance. For example, if a bucket's linked list becomes too long, Java 8 will replace it with a balanced tree (TreeNode).

Explain the usage of **Collections** utility class

The Collections utility class in Java provides an API for working with classes in the Collections framework, such as Lists, Maps, Sets, and Collections. It contains static methods that perform operations on or return collections.

Here are some ways the Collections utility class can be used:

- **Perform common tasks**

The Collections class provides static utility methods that simplify common tasks like sorting, searching, and reversing collections.

- **Create immutable views**

Methods like unmodifiableList or unmodifiableSet can be used to create an immutable view of collections, which can enhance security.

- **Return thread-safe versions**

Methods like synchronizedList or synchronizedSet can be used to return a thread-safe version of the collection, ensuring concurrent access.

- **Shuffle a list**

The shuffle method randomly permutes a specified list using a specified source of randomness.